

(19) World Intellectual Property Organization
International Bureau



(43) International Publication Date
24 April 2003 (24.04.2003)

PCT

(10) International Publication Number
WO 03/034209 A1

(51) International Patent Classification⁷: **G06F 9/44**

Wellington (NZ). **ELLIOTT, Joshua** [NZ/NZ]; Flat 2, 131 Manners Street, Wellington (NZ).

(21) International Application Number: PCT/NZ01/00232

(22) International Filing Date: 19 October 2001 (19.10.2001)

(74) Agents: **HAWKINS, Michael, Howard** et al.; Baldwin Shelston Waters, P.O. Box 852, Wellington 6001 (NZ).

(25) Filing Language: English

(26) Publication Language: English

(71) Applicant (for all designated States except US): **UNISYS CORPORATION** [US/US]; Unisys Way, Blue Bell, PA 19424-0001 (US).

(81) Designated States (*national*): AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, DE, DK, DM, DZ, EC, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NO, NZ, PH, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, TZ, UA, UG, US, UZ, VN, YU, ZA, ZW.

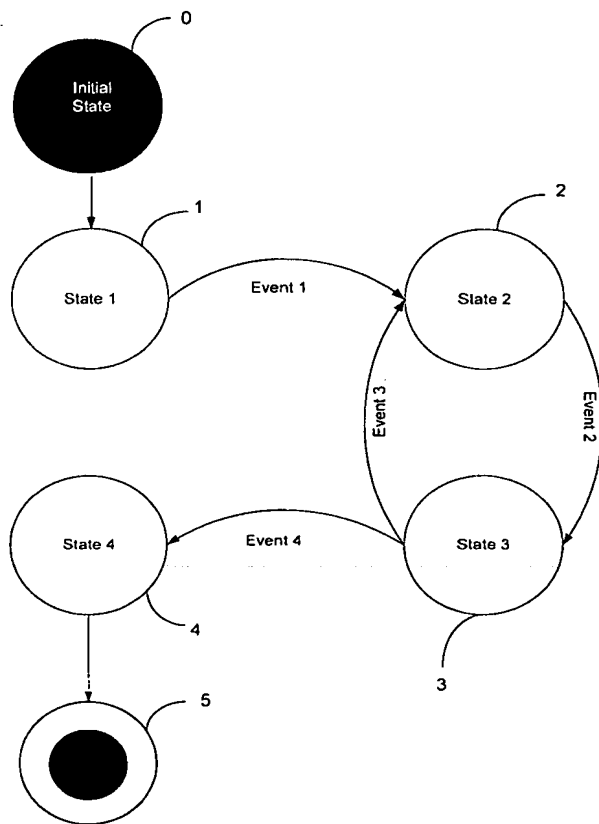
(72) Inventors; and

(75) Inventors/Applicants (for US only): **HOGAN, Timothy, James** [NZ/NZ]; 305 Horokiwi Road, Wellington (NZ). **CARTER, Paul** [NZ/NZ]; 66 Raroa Road, Kelburn,

(84) Designated States (*regional*): ARIPO patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE,

[Continued on next page]

(54) Title: A STATE MACHINE PROGRAMMING LAUGUAGE, A METHOD OF COMPUTER PROGRAMMING AND A DATA PROCESSING SYSTEM IMPLEMENTING THE SAME



(57) Abstract: A state machine language having a syntax requiring each state to be uniquely named and having associated state definition information including: (i) the definition of each action to be executed upon transition to that state; and (ii) the definition of each event which will cause transition to another state and the name of the next state to which operation will progress.

WO 03/034209 A1



IT, LU, MC, NL, PT, SE, TR), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

Published:

— *with international search report*

**A STATE MACHINE PROGRAMMING LANGUAGE, A METHOD OF
COMPUTER PROGRAMMING AND A DATA PROCESSING SYSTEM
IMPLEMENTING THE SAME**

Field of the Invention

The present invention relates to a state machine programming language and its implementation. The state machine programming language of the invention enables direct programming in the state machine language to implement a state machine.

Background to the Invention

A state machine specifies the sequences of states that an object or an interaction goes through during its lifetime in response to events, together with its responses to those events. State machine models are particularly suited for modelling certain systems, such as automated voice-prompting telephone answering systems, traffic light systems, electronic circuits etc.

Currently available computer languages are generally purely procedurally based (eg. C/C + +, Pascal, Java, Basic). To implement a state tree definition a programmer must create a state machine that executes the states. This can be time consuming utilising currently available computer languages and the solution may not be reusable.

Certain applications have been developed for creating state machines but they are generally limited to particular applications and/or complex and/or time consuming to implement.

US5485600 discloses a visual modelling system for defining

relationships between objects. The relationships may be defined
utilising a state table. A user is required to set up the environment for
each state table, edit the state table for the particular application and
then generate a program utilising information from the state table. The
5 state table is utilised as a tool to generate a program utilising a pre-
existing language. There is no suggestion that programming may be
effected directly via a state machine programming language. The
programming method disclosed in this patent is time consuming and
laborious to implement and does not facilitate the reuse of
10 programming code.

Summary of the Invention

15 It is an object of the present invention to provide a state machine
programming language that allows a programmer to program directly
according to a state machine model that clearly defines the structure
of a state machine or to at least provide the public with a useful
choice.

20 According to a first aspect of the invention there is provided a method
of computer programming utilising a state machine programming
language including the steps of:

25 defining a plurality of states according to the syntax of the state
machine programming language, each state having state definition
information including:

- 30 i. a definition of each action to be executed upon transition to that
state; and
- ii. a definition of each event which will cause transition to another
state and the name of the next state to which operation will
progress.

According to a further aspect there is provided a data processing system including:

a storage device containing state information according to the syntax of a state machine programming language including:

- 5 definitions for a plurality of states in which each state definition includes:
- i. a definition of each action to be executed upon transition to that state; and
 - ii. a definition of each event which will cause transition to another
- 10 state and the next state to which operation will progress; and
- a processor which executes the actions in accordance with the state information for the current state and effects state transitions in response to event information.

15 According to a further aspect there is provided a computer programmed to operate according to a state based programming language wherein to create a program the computer requires a user to enter state information according to a required syntax for each state including:

- 20 i. a state name;
- ii. actions to be executed upon transition to the state; and
 - iii. each event which will cause transition to another state and the name of the next state.

25 Brief Description of the Drawings

The invention will now be described by way of example with reference to the accompanying drawings in which:

30 Figure 1: shows a simple state machine.

Figure 2: shows a computer system suitable for executing the state machine programming language of the invention.

Figure 3: shows a state machine for implementing an automated voice-prompting telephone answering system.

5 Detailed Description of the Best Mode for Carrying Out the Invention

Referring to Figure 1 a graphical representation of a state machine is shown. From an initial State 0 operation may progress to a first State 1. Whilst in State 1 certain actions may be executed. Upon the
10 occurrence of an Event 1 operation may progress to State 2 where another set of actions may be performed. Upon the occurrence of Event 2 operation may proceed to State 3 where a further set of actions may be executed. Upon the occurrence of Event 3 operation may return to State 2 or, upon occurrence of Event 4 operation may
15 proceed to State 4 where a further set of actions may be executed. Operation may then terminate at a final State 5.

Although state models have been used to model sequential processes, state diagrams and tables have merely been utilised as a tool to
20 develop a resulting program.

According to the present invention a state machine language has been developed whereby a programmer may structure a program in the state machine language in a manner reflecting the logical operation within a
25 state machine diagram.

According to the invention a machine based programming language is provided which separates state definitions from procedural code and treats the procedural code as an adjunct to the main state handling
30 code. This allows a programmer to clearly define the structure of a state machine and simplifies the writing and maintaining of programs to implement state machines.

According to the state machine language of the invention each state is given an unique state name by the programmer; the actions to be executed whilst the program is in that state are defined; and the events causing the program to progress to a new state along with the new state are also defined. The format for defining each state is as below:

```
State Name
    actions
10         Function call 1
           Function call 2
           Function call n
    events
15         event 1: next state 1
           event 2: next state 2
           event n: next state n
end
```

One or more action may be defined for each state. A number of states may execute the same action. One or more event may trigger a transition to a new state. Different events may trigger transitions to different states.

Each state is preferably defined within a block of code. Each state definition is preferably separated from each other state definition by a blank line.

The actions defined within each state definition may be common functions which may be defined within a "functions" definition. This may be conveniently provided at the end of the state definitions.

The events that may trigger a state transition may be inputs from input devices, the outputs of function calls etc. Any function may execute

any other function which will, when executed, return to the calling parent function. A function is represented by its name, its interface and a set of instructions defining the functions operation as follows:

- 5 Function name
- Interface
- Instructions defining the function's operation

10 Within a state machine it may be desirable to identify a portion of the state tree as a sub-branch. When operation proceeds to the sub-branch, operation may proceed according to the sub-branch state definition until operation proceeds to the final state of the sub-branch, whereupon operation then returns to the parent state tree definition. This approach may be applied to any sub-branch depth. Accordingly,

15 for example, the state diagram shown in figure 1 may represent a sub branch of a larger state diagram. The section of definitions for states 1-4 may be identified as a sub-branch. Such a sub-branch may be called as an action for a state of the parent state tree definition. This may enable commonly used sub-branches to be defined once and to be

20 used by any parent state. This allows state definitions to be compact, simple, easy to understand and better structured. This approach is analogous to a parent function calling another child function that returns back to the calling parent when it is finished.

25 Referring now to figures 2 and 3 a computer for implementing the state machine language of the invention is described in conjunction with an automated voice prompting telephone answering system application by way of example only.

30 Referring to figure 2 the computer may be of standard architecture including a microprocessor 6 which receives input from an input device 7 such as a keyboard. RAM 8 provides temporary data storage and hard disk drive 9 provides permanent storage. An operating system

and the state machine language program may be stored on hard disk 9 and loaded into RAM during operation. Processor 6 outputs graphics information to graphics driver 10 which drives display 11. Typically a user will type in a code which will be displayed by display 11, stored on hard disk drive 9 and executed by processor 6.

Figure 3 illustrates a state machine diagram for a telephone voice prompting system for accessing departments of a business. From Initial State 20 operation proceeds to state 21 where a caller is asked whether operator assistance is required. If "0" is pressed operation proceeds to State 22 and the call is connected to an operator. If a "1" is pushed operation proceeds to State 23. If "1" is selected the call is connected to gardening in State 24. If key "2" is selected operation proceeds to step 25 and the call is connected to the service desk. Once the required functions are executed in states 22, 24 and 25 operation proceeds to final state 26. If any invalid key is selected in State 23 operation proceeds to step 27 and returns to State 23.

The following is a possible implementation of such a system utilising the state machine language of the invention:

// The first state is always the start state for the machine

State AskIfOperatorRequired

```

25         actions
                SendPrompt ("Press 0 to contact an operator or 1
                        for a list of departments");
        event
        "0" :      ConnectCallToOperator;
30        "1" :      SelectDepartment;
end

```

State ConnectCallToOperator

```

        actions
35                SendMessage ("Connecting to the operator");

```

```

                                ConnectCall ( Operator );
                                event
                                default:    FINAL_STATE;
end
5
State SelectDepartment
    actions
        SendPrompt ("Please select the number of the
10        department you are interested in or * to talk to the
        operator");
        event
            '1' : ConnectCallToGardening;
            '2': ConnectCallToServiceDesk;
            '*': ConnectCallToOperator;
15        default: InvalidDepartment;
    end

State ConnectCallToGardening
    actions
20        SendMessage ("Connecting to the Gardening
        department");
        ConnectCall ( Gardening);
        event
            default: FINAL_STATE;
25    end

State ConnectCallToServiceDesk
    actions
        SendMessage ("Connecting to the Service
30    Desk");
        ConnectCall (ServiceDesk);
        event
            default: FINAL_STATE;
    end
35
State InvalidDepartment
    actions
```

```

                SendMessage ("Sorry, no such department
number");
                Desk");
        event
5           default: SelectDepartment;
end

```

FUNCTIONS

```

10      function ConnectCall (department)
        {
            ExtensionNumber = lookup (department);
            RouteVoiceCall (ExtensionNumber);
15      }

        function SendPrompt (prompt)
        {
            VoiceContent = ConvertToVoice (prompt);
20      SendVoice (VoiceContent);
            Key = GetUserResponse();
            Return key;
        }

25      function SendMessage (message)
        {
            VoiceContent = ConvertToVoice (message);
            SendVoice (Voice Content);
30      }

```

In this example the first state "AskIfOperatorRequired" is the initial state. The action definition for this state includes the "SendPrompt" function. This function is defined at the end of the programme definition which instructs the content to be converted to voice. The

35 bracketed content "press "0" to contact an operator or "1" for a list of departments" will thus be sent as a voice message to the caller. In the

event definition the event of pressing key "0" results in a transition to state ConnectCallToOperator and pressing "1" results in a transition to the state SelectDepartment. The other state definitions are likewise defined by their names, actions and the events (each event having an associated next state).

The functions are defined by their name (eg. "ConnectCall"), interface (eg. Department) and instructions (eg. ExtensionNumber = lookup (department); RouteVoiceCall (ExtensionNumber)).

By way of comparison the following code is an example of an implementation of the state machine diagram of figure 3 written in a procedural programming language.

```

15  program VoicePrompter ( )
    begin
        currentState = AskIfOperatorRequired
        While currentState != FINAL_STATE do
            begin
20         key = GetUserKey( )
            select CurrentState of
                case AskIfOperatorRequired :
                    {
                        key = SendPrompt (``Press 0
25         to contact an operator or 1 for a list of departments``);
                        if key = `0' then
                            CurrentState =
                                ConnectCallToOperator;
                        Else
30         CurrentState =
                                SelectDepartment;
                    }
                case ConnectCallToOperator:
                    {
35         SendMessage (``Connecting to the
                                operator``);

```

```

        CurrentState = FINAL_STATE;
        ConnectCall ( Operator )
    }
    case SelectDepartment:
5      }

        Key = Sendprompt ( ``Please select the
number of the department you are interested in or * to talk to
the operator``);

        If key = '1' then
10      CurrentState =
ConnectCallToGardening;
        Else
        If key = '2' then
            CurrentState =
15      ConnectCallToServiceDesk;
        Else
        If key = '*' then
            CurrentState =
20      ConnectCallToOperator;
        Else
            CurrentState =
Invalid Department;
        }
        case ConnectCallToGardening:
25      {
            SendMessage ("Connecting to the
Gardening department");
            CurrentState = FINAL_STATE;
            ConnectCall ( Gardening );
30      }
        case ConnectCallToServiceDesk:
        {
            SendMessage ("Connecting to the Service
Desk");
35      CurrentState = FINAL_STATE;
            ConnectCall (ServiceDesk);
        }
    }
```

```

                                case InvalidDepartment:
                                {
                                    SendMessage ("Sorry, no such
5                                department number");
                                CurrentState =
                                    SelectDepartment;
                                }
                                end case
                                end while
10                               end program
```

```

                                function ConnectCall (department)
                                {
15                               ExtensionNumber = lookup (department);
                                    RouteVoiceCall (ExtentionNumber);
                                }
```

```

                                function SendPrompt (prompt)
                                {
20                               VoiceContent = ConvertToVoice (prompt);
                                    SendVoice (VoiceContent);
                                    Key = GetUserResponser ();
25                               Return (key);
                                }
```

```

                                function SendMessage (message)
30                               {
                                    VoiceConent = ConvertToVoice (message);
                                    SendVoice (VoiceContent);
                                }
```

35 It will be noted that the state information is buried amongst procedural control statements and the states, actions and transitions of the state machines are difficult to discern.

It will thus be seen that the invention provides a state machine language which allows a programmer to clearly define the structure of a state machine, and thus facilitate the writing and maintaining of programs implementing state machines. The syntax of the state machine language of the invention allows states and transitions to be explicitly defined, making the state machine structure readily discernible. This greatly simplifies writing and maintaining programs written in the state machine language of the invention.

5

10

CLAIMS:

1. A method of computer programming utilising a state machine programming language including the steps of:

5

defining a plurality of states according to the syntax of the state machine programming language, each state having state definition information including:

10

- i) a definition of each action to be executed upon transition to that state; and

- ii) a definition of each event which will cause transition to another state and the name of the next state to which operation will progress.

15

2. A method as claimed in claim 1 wherein each state name, its associated actions and events are grouped together.

20

3. A method as claimed in claim 1 or claim 2 wherein functions called by actions are defined separately from the state definitions.

4. A method as claimed in claim 3 wherein the functions are defined following the state definitions.

25

5. A method as claimed in any proceeding claim wherein multiple actions are executed upon transition to a state.

30

6. A method as claimed in any proceeding claim wherein an action is executed by a plurality of states.

7. A method as claimed in any one of the proceeding claims wherein at least two events are defined for a state, each causing a transition to a different state.
- 5 8. A method as claimed in any one of the proceeding claims wherein a state transition is contingent upon multiple events.
9. A method as claimed in any one of proceeding claims wherein one or more state includes one or more sub states.
- 10 10. A method as claimed in claim 9 wherein the sub states are defined in the manner defined in any one claims 1 to 8.
11. A method as claimed in any one of the proceeding claims wherein
15 an event is a default event, being any or no action.
12. A method as claimed in one of the preceding claims wherein the state definitions are entered via a user input device of a computer.
- 20 13. A method as claimed in one of the preceding claims wherein the state definitions are stored in computer readable media.
14. A computer program produced according to the method of any one of the preceding claims.
- 25 15. A computer readable medium embodying a computer program as claimed in claim 14.
16. A computer programmed to operate according to the computer
30 program of claim 14 or claim 15.

17.A data processing system including:

a storage device containing state information according to the
syntax of a state machine programming language including:
5 definitions for a plurality of states in which each state definition
includes:

- i) a definition of each action to be executed upon transition
to that state; and
- 10 ii) a definition of each event which will cause transition to
another state and the next state to which operation will
progress; and
- 15 iii) a processor which executes the actions in accordance
with the state information for the current state and
effects state transitions in response to event
information.

18.A system as claimed and 17 wherein the storage device contains
state information in accordance with any one of claims 1 to 13 at.

20 19.A system as claimed in claimed 17 or claim 18 wherein the event
information is input data.

20.A system as claimed in claimed 17 or claim 18 wherein the event
information is the output of one or more actions.

25 21.A system as claimed in any one of claims 17 to 20 wherein the
system is an automated voice-prompting telephone answering
system.

30 22.A computer programmed to operate according to a state based
programming language wherein to create a program the computer
requires a user to enter state information according to a required
syntax for each state including:

- i) a state name;
- ii) actions to be executed upon transition to the state; and
- iii) each event which will cause transition to another state and the name of the next state.

5

23. A computer as claimed in claim 22 including a display device which displays state information for each state within a display sub region.

10

24. A computer as claimed in claim 23 wherein the state information for each state is displayed as a separate block of text.

25. A computer as claimed in claim 23 or 24 wherein functions called by actions are displayed in a separate function block.

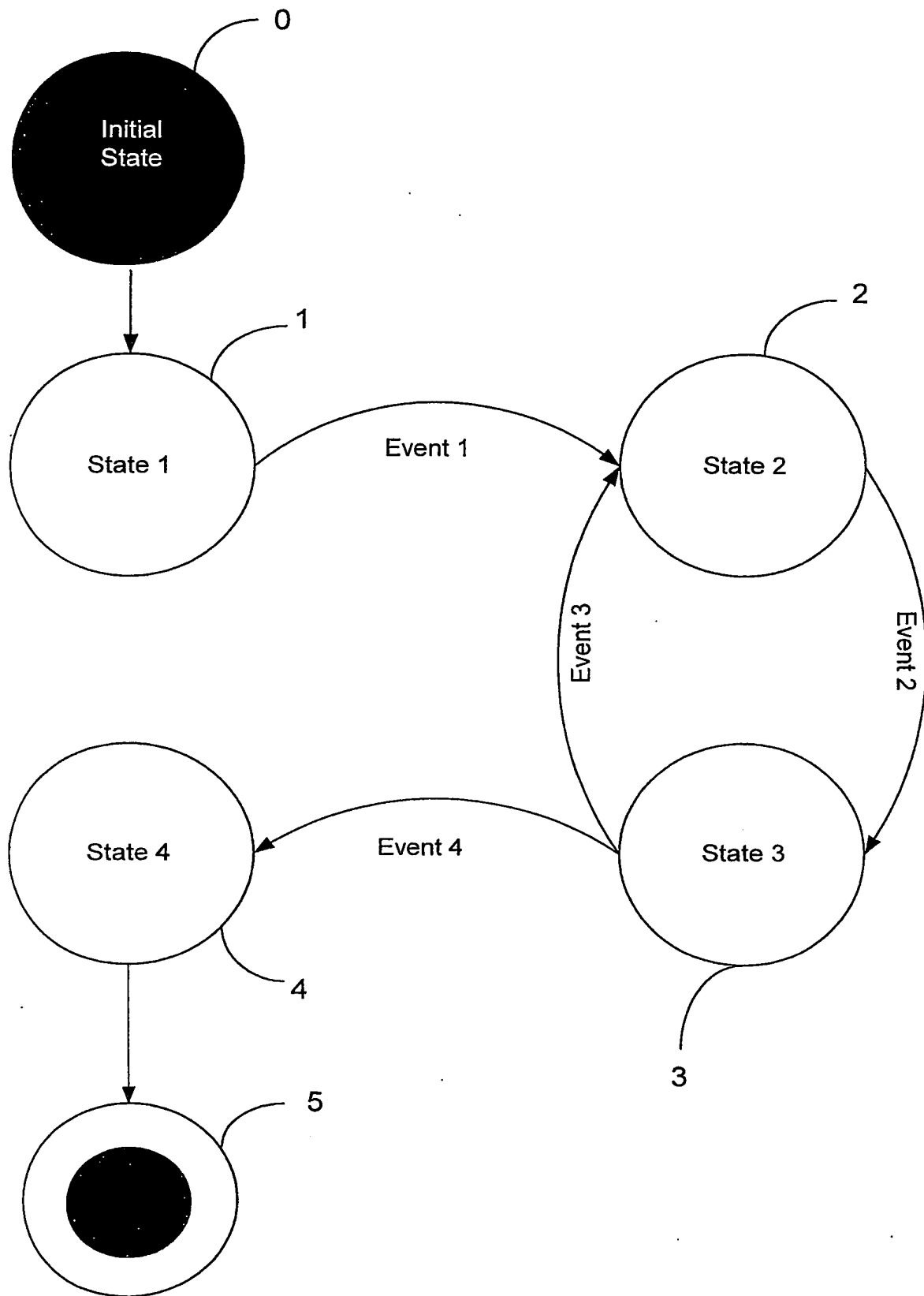
15

26. A computer as claimed in any one of claims 22 to 25 which executes the defined actions for a state upon transition to that state and effects transition to a next state upon the occurrence of a specified event.

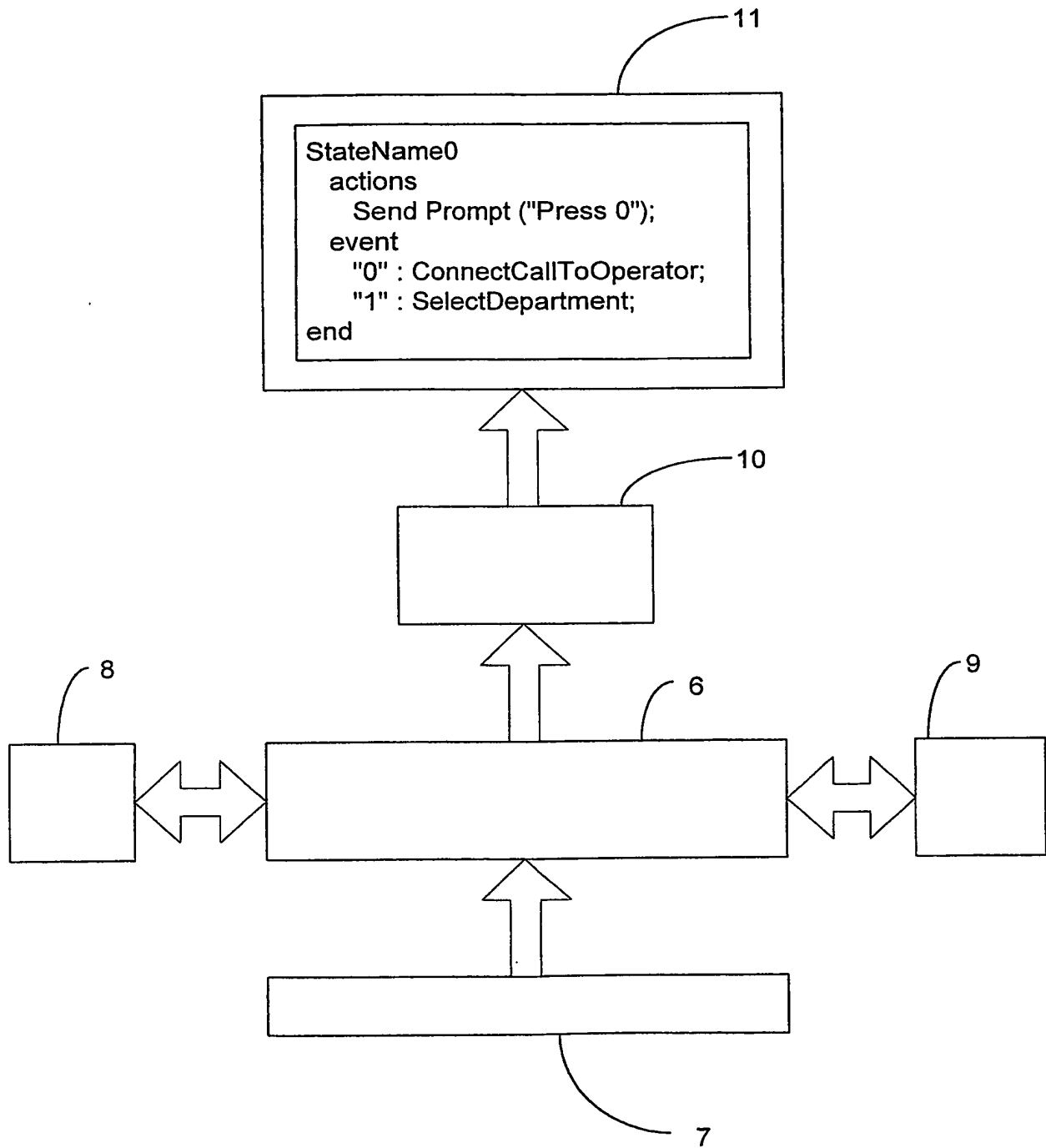
20

27. A computer as claimed in claim 26 wherein the computer executes the actions by calling functions from the function block.

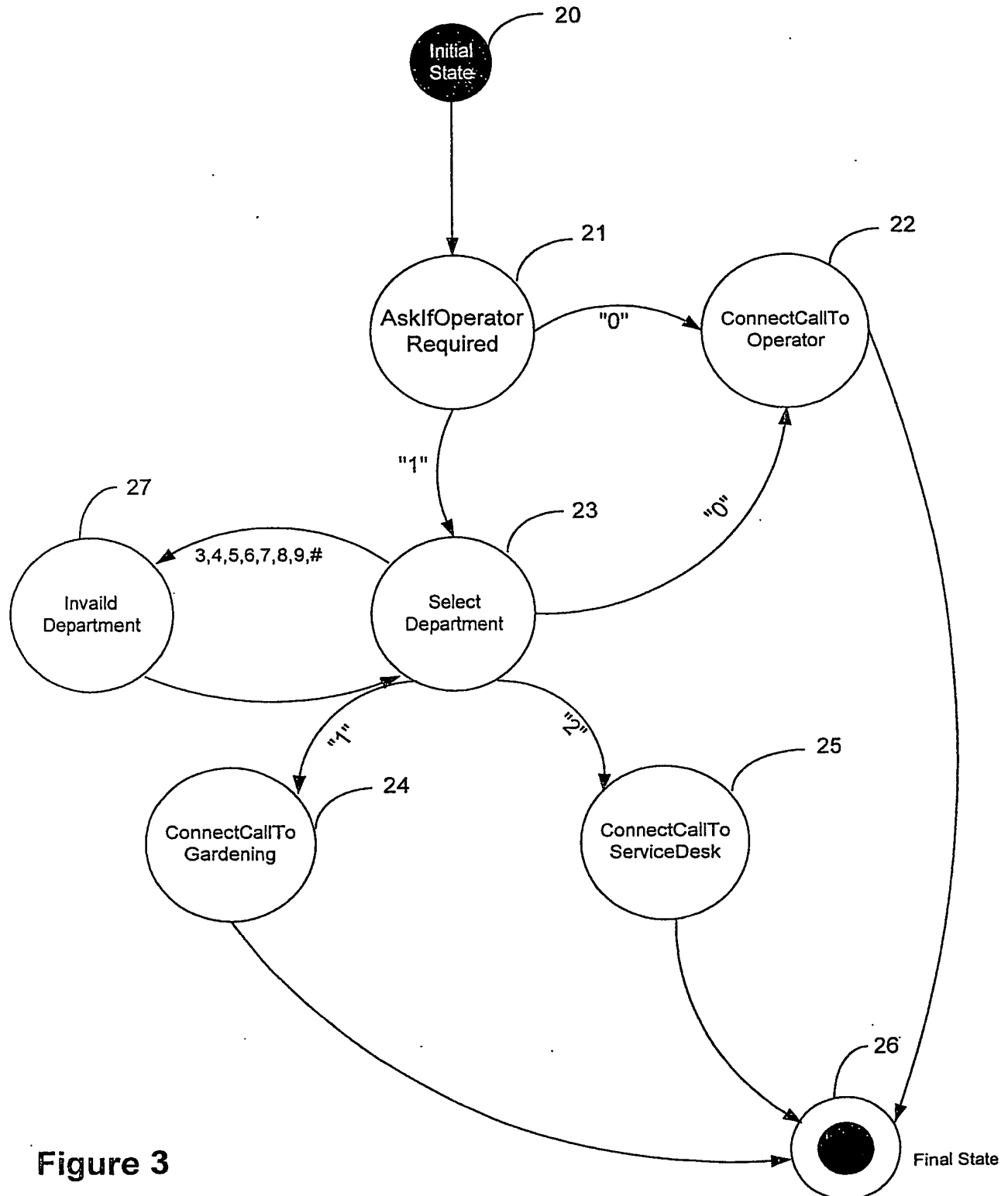
1/3

**Figure 1**

2/3

**Figure 2**

3/3

**Figure 3**

INTERNATIONAL SEARCH REPORT

International application No.

PCT/NZ01/00232

A. CLASSIFICATION OF SUBJECT MATTERInt. Cl. ⁷: G06F 9/44

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)

WPAT,IEEE (state machine language)

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	US 5 542 070 (LeBlanc et al.) 30 July 1996	1,13-16
Y	Abstract, column 3 line 52 to column 5 line 20, claim 2	17-20,22,27
Y	WO 00/78022 (Telstra R&D Management Pty. Ltd.) 21 December 2000	21
	Abstract	
Y	US 5 920 718 (Uczekaj et al.) 6 July 1999	17-20,22,27
	Abstract, figures, claims	

☒ Further documents are listed in the continuation of Box C
 ☒ See patent family annex

* Special categories of cited documents:	
"A" document defining the general state of the art which is not considered to be of particular relevance	"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention
"E" earlier application or patent but published on or after the international filing date	"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone
"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)	"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art
"O" document referring to an oral disclosure, use, exhibition or other means	"&" document member of the same patent family
"P" document published prior to the international filing date but later than the priority date claimed	

Date of the actual completion of the international search

14 December 2001

Date of mailing of the international search report

20 DEC 2001

Name and mailing address of the ISA/AU

AUSTRALIAN PATENT OFFICE
 PO BOX 200, WODEN ACT 2606, AUSTRALIA
 E-mail address: pct@ipaustalia.gov.au
 Facsimile No. (02) 6285 3929

Authorized officer

DALE E. SIVER

Telephone No : (02) 6283 2196

INTERNATIONAL SEARCH REPORT

International application No.

PCT/NZ01/00232

C (Continuation). DOCUMENTS CONSIDERED TO BE RELEVANT		
Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
Y	"A Tool for Practical Reasoning about State Machine Designs" (Cant et al.) Proceedings of the 1996 Australian Software Engineering Conference ASWEC -'96 Whole document, esp. Sections 1.1 and 2	1,17,22
A	US 5 937 181 (Godefroid et al.) 10 August 1999 Abstract, Summary	1

INTERNATIONAL SEARCH REPORT
Information on patent family members

International application No.
PCT/NZ01/00232

This Annex lists the known "A" publication level patent family members relating to the patent documents cited in the above-mentioned international search report. The Australian Patent Office is in no way liable for these particulars which are merely given for the purpose of information.

Patent Document Cited in Search Report		Patent Family Member	
US	5542070	CA	2122182
WO	2000/78022	AU	2000/50545
US	5920718	NO	MEMBERS
US	5937181	NO	MEMBERS
END OF ANNEX			